FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Data Integration Solution in an Heterogeneous Environment

Pedro Manuel dos Santos Rocha

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Gabriel David

Second Supervisor: Daniel Costa

June 27, 2017

Data Integration Solution in an Heterogeneous Environment

Pedro Manuel dos Santos Rocha

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Rui Carlos Camacho de Sousa Ferreira da Silva
External Examiner: Paulo Jorge Machado Oliveira
Supervisor: Gabriel de Sousa Torcato David

June 27, 2017

Abstract

Over the last few years there has been an increase in the attention given to both data collection and knowledge extraction. Recent developments in data storage, distributed systems and parallelization made the analysis of vast amounts of data more straightforward. However, whilst processing large quantities of information has been made simpler there are still some problems that need to be addressed. One of these problems resides in the clean-up of the data collected, meaning the transformation of the information collected into a more useful format from where knowledge can be extracted.

Usually this problem is addressed by developing a solution, using data integration tools, on a case by case basis that has no power of generalization. As expected, this type of solution works well in an environment where the data is well known and with a fixed structure, but if there are changes in the initial or the final structure of the information there needs to be an adjustment made to the solution. This brings added complexity that can cause the application to become increasingly difficult to maintain and add new features.

The solution that is analyzed throughout this dissertation is the creation of a configurable application where a user can combine and transform information that originates from different sources. This is made utilizing user-defined configuration documents, so that when a change is made in the system the impact for the data integrator is minimized.

In order to assess the suitability of the solution, a real-world scenario has been used. This scenario is based on an already existing application that collects information from a variety of sources and has the necessity of transforming the collected information into a more useful and stable structure.

Resumo

Ao longo dos últimos anos tem vindo a haver um aumento da atenção dada à recolha de informação e extração de conhecimento. Desenvolvimentos recentes nas áreas de armazenamento de dados, sistemas distribuídos e paralelização fizeram com que a análise de vastas quantidades de informação se tornasse mais direta. No entanto, apesar do processamento de grandes quantidades de informação ser mais simples ainda existem alguns problemas que necessitam ser adereçados. Um destes problemas está localizado na limpeza da informação que é recolhida, ou seja a transformação da informação num formato mais simples para a extração de conhecimento.

Normalmente, este problema é abordado com o desenvolvimento de uma solução específica, utilizando ferramentas de integração de informação, para cada caso, sem a capacidade de ser generalizada. Como esperado, este tipo de solução funciona bem num ambiente onde os dados são bem conhecidos e possuem uma estrutura fixa, mas se existir uma alteração na estrutura inicial ou final da informação é necessário fazer um ajuste à solução. Esta situação traz mais complexidade, o que pode causar um aumento de dificuldade na manutenção e desenvolvimento de novas funcionalidades numa aplicação.

A solução que está a ser analisada nesta dissertação é o desenvolvimento de uma aplicação configurável onde um utilizador pode agregar e transformar informação que origina de fontes de informação diferentes. Esta capacidade é conseguida através da utilização de documentos definidos pelo utilizador para que uma modificação no sistema implique o mínimo de impacto para o integrador de dados.

De forma a melhor avaliar a usabilidade da aplicação foi usado um ambiente real. Este cenário baseia-se numa aplicação que extrai informação de várias fontes de informação e, de momento, necessita de diversos módulos diferentes para fazer a transformação de informação da sua estrutura inicial para uma mais adequada e estável.

Acknowledgements

Firstly I would like to thank both Professor Gabriel David for his support throughout the development of this dissertation and Engineer Daniel Costa for proposing this theme as well as giving me the chance to work on this challenge.

Another praise goes to all the people of Mog Technologies for welcoming me and, in particular, to the members of the Data Science team, Álvaro Ferreira, Luis Araújo and Pavel Alexeenko as well as Pedro Santos for their support for the duration of this dissertation and for the patience they demonstrated during the development period.

A last note to the panelists and members of "Mais Boatos" for the creation of a great environment for the necessary breaks over the course of the work.

Pedro Rocha

"I disapprove of what you say,
but I'll defend to the death your right to say it.'

Evelyn Beatrice Hall

Contents

List of Figures

LIST OF FIGURES

List of Tables

Abbreviations

IBM	International Business Machines
NoSQL	Not only SQL
SQL	Structured Query Language
MXF	Material Exchange Format
GXF	General Exchange Format
IBM	International Business Machines
HDFS	Hadoop Distributed File System
JSON	JavaScript Object Notation
XML	eXtensible Markup Language
IP	Internet Protocol

Chapter 1

Introduction

Nowadays information has become a highly sought after commodity. This change has been brought about by recent developments on both the collection and the analysis of information.

When the Web 2.0 emerged there was a shift in the dynamics of the Internet, instead of users accessing static web pages, users became producers of content. This was brought about by increases in both the quality and availability of internet services and the rise in popularity of social media. Another important factor for this change was the smartphone, which allowed people to access the Internet nearly anywhere and provided a new source of data that could provide useful knowledge. This has led to the creation of new and useful information that companies wanted to extract knowledge from. Even though data analytics have been used by companies for over half a century, there was a new challenge - Big Data. Information was being created at an unprecedented pace and traditional methods used to both analyze and store information were not designed to be able to handle it.

Both the knowledge extraction techniques and the storage methods for information were facing difficulties. Traditionally, knowledge extraction was made by using statistical algorithms that used a relatively small amount of information to make their calculations, thus being unable to handle such vast quantities of information. On the other hand relational databases had been the *de facto* business standard since the 1980s and, although they could theoretically handle the seemingly endless volume of information that needed to be stored, they could not cope with the velocity at which it was being created or the variety of the information that was being created. This necessity led to the creation of new ways to store and analyze information.

Currently, the analysis and storage of large amounts of information has been somewhat trivialized, but some problems are still present. Most of these problems revolve around the preprocessing the data needs in order for knowledge to be extracted from it. One of the reasons for this is the existence of multiple sources of information that store information differently and that are subject to change over time. These variations can create issues if data needs to follow a particular structure in order to be properly analyzed, implying a need to restructure the information. An additional

issue occurs when just the final structure of the information is known, and not both the initial and final data structures. This problem is the focus of this dissertation work.

The proposed solution to this problem is the creation of a configurable application able to map different or varying sources to a common final data structure. This technique has been developed using a real world example, an application where one needs to collect information from a variety of different sources. The information being gathered needs to be transformed in order to be useful. Previously there were several different modules in charge of this transformation, one for each type of data that is being analyzed. The end results of the work developed during this dissertation are compared against the previously used solution.

1.1 Context

This dissertation comes from a need in data science for a way to more easily clean-up data before it can be properly analyzed.

Different concepts can be learned from different data. With this in mind there is a tendency to always collect all of the information available and, before analyzing the data, the unnecessary information is removed and any preprocessing the remaining data needs is made. This process is normally done by developing *ad hoc* solutions that lack any flexibility. While this approach can work on certain environments it is not adequate if there is mutability in the system. This creates the need to develop a solution that can be generalized to work with a variety of data and transformations required.

This dissertation is being developed with MOG Technologies which develops digital media workflow solutions. It offers centralized ingest solutions and MXF development tools to broadcasters, post-production houses, outside broadcasters, mobile studios, and live-feed productions worldwide. Its centralized ingest solutions include mxfSPEEDRAIL, a file-based solution for post-production, playback, file archiving, and multi-channel workflows. The company also provides MXF and GXF tools that enable accelerated development and migration of IT-based multimedia content applications. In addition, it offers workshops, technical training sessions, and on-site support and consultancy services.

This company is developing an application that collects information from a variety of video editing software, which is then used to ease managerial decisions. In order to achieve this goal the data must be processed, by a cataloger, so that irrelevant information is filtered out. This task was previously being executed by a series of modules that needed to be developed specifically for each type of information, figure 1.1. This solution causes a large amount of overhead when there is a change in the system or a new type of data is to be analyzed. This dissertation comes as a way of replacing the cataloger, making it so that there is no longer a need to program different modules for different informations, instead being necessary to replace the configuration documents. This makes the adaptation to new information simpler, alter two documents instead of programming a new module.

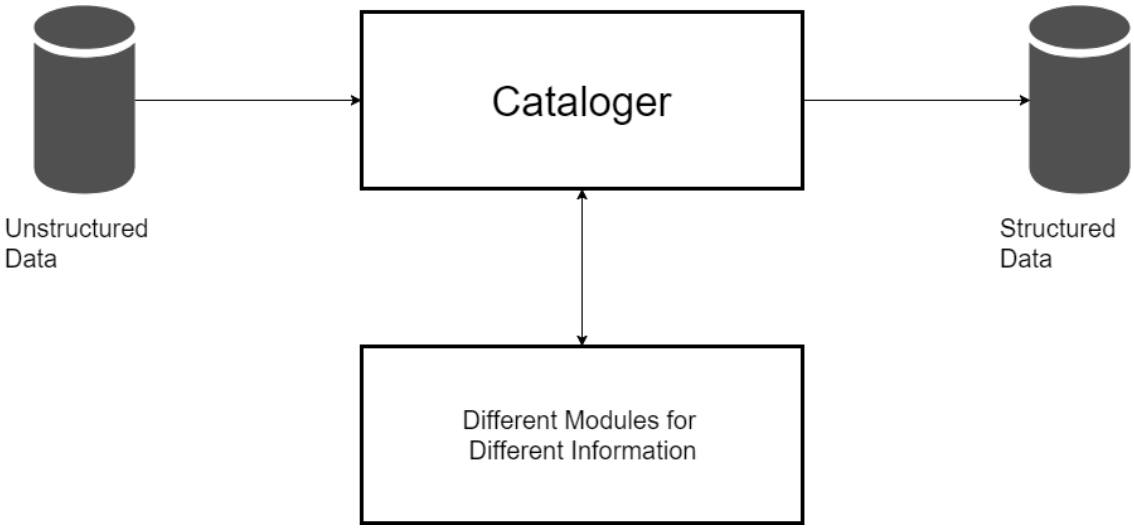

Figure 1.1: Current Application Workflow

1.2 Goals

The motivation for this work comes from an interest in both the areas of database management and data science. This work allies both of the disciplines described above. There is a need to cleanse the data in order to analyze it and it is necessary to allow a user to query the information being retrieved. Allied to these factors is the lack of a current solution to the problem that makes it more enticing.

The final objective of this dissertation is the creation of a configurable application that is able to map data from a category of sources to a defined target data structure. This will be achieved by first analyzing the state of the art and ascertaining the better way to determine correspondences between the information being retrieved from the data sources and the final structure defined by the user.

1.3 Report Organization

The structure of the paper is as follows. In section 2 are presented some of the tools and technologies that are explored over this dissertation. Section 3 explores the technologies and concepts that are relevant to data storage and machine learning. In section 4 the problem being addressed is further explained and the proposed solution is presented. In section 5 the two main steps of the application are described. Finally, the conclusions and future work are drawn in Section 6.

Chapter 2

Tools and Technologies

In this section are presented the concepts and tools that will be used throughout this dissertation.

2.1 Big Data

The attention given to Big Data has increased over the last years due to its increasing importance to business and people's lives. This has been triggered by the unprecedented opportunities that enable data collection, the main one being the widespread of smartphones and an almost permanent connection to the internet.

A problem that arose from this rapid development was the lack of a clear definition of what Big Data is. The first definitions used three V's to describe it: volume, variety and velocity [GH15]. Volume refers to the shear amount of data that is being collected. Since different types of data occupy different amounts of space and the amount of disk space available is steadily increasing over time, it becomes difficult to define a limit from which the data set becomes Big Data. Variety refers to the different types of data that can be used, structured data, semi-structured data and unstructured data, of which only a small portion is structured. This type of distribution is present ever since companies started collecting analytics but only recently has there been techniques able to process this data into useful information. Velocity refers to the speed at which data is created and at which it needs to be analyzed. This means that a stream of data can be used to create real-time analytics, for example recommendation systems in e-commerce use this to create personalized deals to customers.

Even though there is an agreement concerning the three V's there are some that added more to their definition. IBM added a fourth V: veracity [IBM], as seen in figure 2.1, which is used to highlight that there is always a degree of uncertainty to Big Data, particularly when using information that originates from social media. However this data can contain useful information and must be analyzed through methods that take uncertainty into account. SAS added two new factors to their definition of Big Data: variability and complexity [SAS]. The former refers to the

inconsistency of data flows, data is not created at the same rate, there are peaks both daily and seasonally that need to be dealt with, the latter refers to the different sources the data can come from and the difficulties that arise when trying to connect and cleanse the data, despite this it is necessary to make these connections in order to better use information. Oracle added a fourth V to their definition: value [Dij14], with the volume of data that Big Data entails the information that can be extracted from the data is spread over a larger space, this creates an added difficulty that one must decide what data can create value before analyzing it.

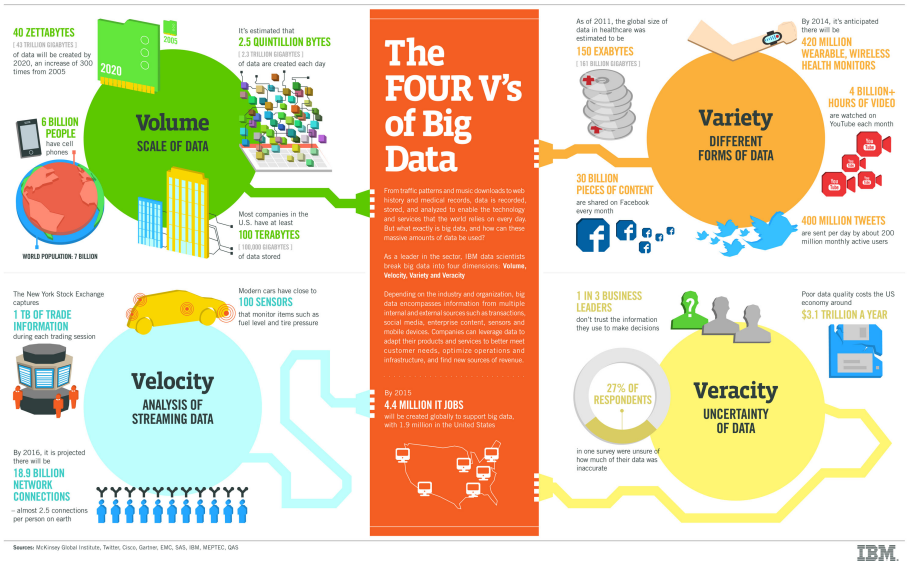

Figure 2.1: IBM's explanation of their 4 V's of Big Data - Available at: http://www.ibmbigdatahub.com/infographic/four-vs-big-data

2.2 Data Science

There is no concrete definition of what data science is, but a consensus has been formed that defines it as a combination of existing disciplines that are used by data scientists such as, computer science, statistics, mathematics, etc. It is used to solve complex data-centric problems, which encompasses all of the steps of data collecting, analysis and modeling.

For many years companies have collected data on all of their activities that could be monitored, but most of that information would end up discarded due to limitations in storage capabilities, or unused. This usually happened because the type of algorithms that were being used were more focused around working with structured data and with a few relevant examples, due to the fact that they were computational and memory intensive. With recent developments in distributed computing allied with the ability to parallelize these algorithms there is now the ability to analyze much more information than before. However these developments were not enough to cope with Big Data and it was required a change in mindset. Traditionally, information was delivered to a single point where it was processed, but with the volume and velocity of Big Data this type

of approach was not adequate. In order to circumvent this, the processing power began being transported to the data, meaning that instead of information being transferred to a central location in order to be analyzed, the computational power was delivered directly to the data by replicating the algorithm used to the different nodes where the information is stored.

2.3 The Tools in Study

As previously described, there was a need to develop new ways to analyze data, that was being created at a great pace. In this section are analyzed the tools that are currently being used by applications to perform the information transformation.

2.3.1 MapReduce

MapReduce is a programming model that was designed to be able to process a very large amount of data in a distributed manner. It was first introduced by Google [DG08]. It is based on two main moments for processing information: a Map phase and a Reduce phase. On the first step, a user-defined Map function is used in order to create a set of key/values, then a, also user-defined, Reduce function is used to merge the values that have the same key. There are two main features that give and edge to MapReduce and they are, the representation of the information in key/value, which can represent a combination of real world situations, and that it was developed from the ground up to be distributed and highly scalable.

The model is based on a single master node, that distributes the work, and several worker nodes, which perform the available tasks. All the data in the system is separated into chunks, generally 16 to 64MB, is replicated several times, then the master node attributes the different Map tasks to the different worker nodes, taking into account the locality of the information. The result of these operations is written to temporary files which are in turn processed by the worker nodes that are in charge of performing the Reduce tasks.

The system is designed so that when a worker node is lost the master can redistribute its tasks to others that have access to the information present in the missing node. In this situation Map tasks are redone from the beginning, unlike Reduce tasks. This situation occurs because the output of a Map task is written in a machine local hard drive, but the result of a Reduce task is written in a global file system. If there is a fail in the master task, there can be two outcomes, the entire task is canceled or, if the master is made to write periodic checkpoints, it can be restarted using the most recent checkpoint state. Another advantage of MapReduce is that all of the task distribution and parallelization is made automatically meaning that a programmer that has little to no experience in these areas can still utilize MapReduce to analyze relevant information.

2.3.2 Apache Hadoop

Apache Hadoop is an open-source framework that was inspired by the Google File System and MapReduce [had]. It makes use of its implementation of MapReduce and HDFS to analyze large

quantities of information. HDFS stores the information that is going to be analyzed by dividing it in chunks and spreading them across the different nodes using a degree of redundancy. Hadoop has been implemented using Java but it provides a service that allows most languages to be used to define the Map and Reduce functions. Besides being developed with cross-functionality in mind its popularity is also bolstered by the additional modules that increase its capabilities. Some of these are HBase, which provides similar functionalities to HDFS and Hadoop as Google's BigTable to Google distributed file system, a highly-scalable, distributed, versioned, non-relational database, and Hive, which is focused on performing traditional data warehousing tasks on data that is being stored using HDFS or HBase.

The main advantage of Hadoop is that it can be used with minimal knowledge about distributed systems and parallelization, as this is handled by the framework. The variety of modules that are available can also extend its usability to more problems.

2.3.3 Apache Avro

Apache Avro has been developed as a part of the Apache Hadoop project [avr]. It functions as a data serialization and remote procedure call tool. Its main use is within Apache Hadoop as a way to serialize persistent information, as a means of communication between the various nodes in the system or to communicate between client applications and Hadoop services. There are two techniques that can be used to store information, a compact binary format or a JSON format.

An advantage of Avro is that while it is necessary to define a schema when a file is being written the same is not applicable when reading a file. This originates from the fact that when a read operation is performed the information schema is supplied. Between writing and reading there can be changes to the data schema, Avro is prepared to handle some of the incongruities that originate from such a case. In order for this to be possible there must be a conscious effort from the user to maintain the name of the attributes or to define default values for the values that are only present on the reader schema. Values that are only present in the writer schema are ignored during the reading task.

2.3.4 Apache Kafka

Apache Kafka was developed as a distributed system for the handling of streams [kafa]. The main difference of Apache Kafka is the structured commit log of updates, meaning that a producer of data can send records that are then appended to this log, which can be later sharded [kafb]. This adaptation means that a series of consumers can all access the data without significant delays and the existence of a series of logs creates a way of maintaining the persistence of data over the course of time.

The main advantage of Apache Kafka is its horizontal scalability, meaning that many applications can be maintained, even if there is a need to create shards that are separated geographically, and allowing the creation of more applications to handle data streams that might have been left

unanalyzed. This capability is enhanced by the fact that all of the data streams are naturally multi-subscriber, meaning that basically no extra work is needed to support a multitude of applications accessing the same data.

2.3.5 Apache Spark

Apache Spark is an open-source data processing engine, developed with a great focus on speed and powerful analytics [spa]. The foundation of this project is Spark Core which provides the base functionalities that are required by applications, such as I/O operations and scheduling, these functionalities are accessible through an API, for JAVA, Scala, Python and R. On top of this driver, there are several modules that add more functionality to Apache Spark.

Spark SQL handles structured information and implements the DataFrames abstraction, that creates a distributed query engine. This module is integrated in the Spark ecosystem meaning is can be used in combination with the other modules. Another module is Spark Streaming, which gives the ability of handling data streams not being limited to batch data.

There are also modules that are focused on handling and transforming graph structured data, GraphX, and making a large array of machine learning algorithms available to process the data, MLlib.

Chapter 3

Data Storage

3.1 Data Sources

Another of the core concepts of big Data is variety, which can represent not only the different types of data that can be present but also the variety of data sources that can be present in a system. This section of the document illustrates the different types of data sources that are currently being used.

3.1.1 Relational Databases

In these types of databases, information is stored using tables, column and rows, where a table represents an entity, e.g. client, a row represents an instance, e.g. Client 1, and columns represent each of the attributes, e.g. name, address, ID. In order to keep the information consistent each row has a unique identifier, key, and there can be no duplicate rows. Depending on the different types of relations that can be established among the different tables they can be represented by using an auxiliary table or by storing the relevant key in the related row. This type of structure allows for the storage information with varying degrees of complexity with relative ease and it can be altered to accommodate any changes in the data that is being stored. The main problem with this types of databases is that, whilst all types of information can be stored, the many rules that have to be followed and checked mean that there is a difficulty in handling a big flow of write or read/write requests. This means that, until recent developments, these types of databases could not handle the velocity that Big Data needs. Another problem arose from the fact that relational databases must follow a strict schema and the data being collected can have a varying structure.

3.1.2 NoSQL Databases

The web 2.0 transfigured the way internet content was being consumed, meaning users changed from partaking in fixed content that was available in mostly static web pages, to becoming themselves creators of content with the advent of social media platforms. This brought about the neces-

sity to handle a huge amount of data, which was beyond the capabilities of traditional relational databases. Such a requirement gave rise to the development of databases that did not depend on the relational model, NoSQL databases. These databases have existed since the 1960's but only recently reached the spotlight due to their capacity for handling large amounts of semi-structured data. This comes from the fact that NoSQL databases have been developed to be horizontally scalable, which made possible the parallelization of their functions. Another design decision that copes with velocity is that these databases have eventual consistency, meaning that when a read operation is made the information retrieved might not be the most recent in the system. this happens because when something is written in the system there is no requirement to halt read operations to maintain the information always consistent.

There are multiple types of NoSQL databases that have different ways to store information.

3.1.2.1 Column-oriented

This type of databases uses columns to store information but, unlike relational databases, those are organized in a more relaxed fashion. Instead of belonging in a row, they can belong to a column family that is similar to a row in traditional databases, but all families can have a different structure from each other.

3.1.2.2 Document-oriented

Being document-oriented means that the information is stored in documents and not using tables like in a relational database. Usually these documents have a specific encoding, e.g. JSON, XML, but it can also use binary files, e.g. PDF. One of the most contrasting points between these and relational databases is that instead of the information being spread across multiple tables all of the information about an instance is stored in a single document. Another point is the information in the documents is schema-less, meaning that each one can have different information and structure from each other.

3.1.2.3 Key-value

In this type of databases information is stored using a structure known as dictionary or hash. This structure is used to store arrays of information, and utilizes a unique value, a key, to identify the data. All information that is stored is treated in the same way regardless of its type, which enables the user to store data that does not share the same structure. However this means that there is also the inability to perform optimizations to the information store. Before the advent of cloud computing this last factor was one of the responsible for the weak adoption of these databases.

3.1.2.4 Graph

Graph databases use graph structures to store information, they utilize nodes, edges and properties. Nodes store the information about an instance, edges represent the connections between the nodes

and properties are used to store extra information about the nodes. This type of structure is usually used when the relationships between the data is equally, or more, important as the data itself. Although this structure can also be stored using a relational database, they are more commonly associated with non relational implementations.

3.1.2.5 Multi-model

Even though all of them are different, all of the above databases utilize a single data model, document-oriented databases only store documents, and so on, multi-model databases can support multiple models. These are particularly useful if an application has many different facets that are better suited to different data models, this comes at the cost of higher complexity in applications.

Chapter 4

Unstructured Data Mapping

In this section will be presented the current problem with the mapping of unstructured data, the solution proposed and the prototype that has been implemented. The proposed validation techniques will also be discussed.

4.1 The Problem

With more and more information being gathered each day, there comes a need to best analyze this data in order to extract the maximum amount of knowledge from it. More often than not, the data being collected is not prepared to be analyzed. This means that there is a need to alter the data in some way, a step that is designated as data cleaning. This process can be used to eliminate information that is redundant, remove values that are not important to what is being analyzed, restructuring the data collected or combining different attributes in order to create something more significant.

This originates from the fact that different knowledge can be extracted from different information, which means that there is always an attempt to collect as much as possible of all of the information that can be collected from the system. Usually, this represents a significant step in the data analysis problem. There is a need to comprehend the problem and the data very well in order to make informed decisions about what must be done for the knowledge to be correctly extracted from the information available.

The focus of this dissertation work will be on the data cleaning process. Currently there isn't a generic process for the mapping of information that originates from unstructured data sources. This creates a need to create a more flexible solution for performing the restructuring of said information. When the information shares a single known schema this is made simpler, however that is not normally the case. In a real world scenario information is often collected from a variety of sources, with a format that is not useful when it comes to data processing, creating the need to transform the information into a structure that is more useful.

When dealing with a variety of data sources, this problem can become a real obstacle when all of them have a unique way to store the information that requires some kind of transformation. Normally the solution to this situation is to create different applications that handle the various structures that the data can have. This solution may be adequate if the system is well known and there are no changes to the data structures. However, in a real situation this is hard to guarantee which means that every time there is a change to the structure of the information, either initial or final, it is necessary to alter all of the modules that are involved in transforming said information. With the relevance that NoSQL databases got over the last few years a new difficulty arose, the data source from which the information is being retrieved could have no general schema. This means that the information must be transformed without knowledge of the structure of the information. An additional problem that needs to be addressed is that the information that might be relevant to a particular system can be spread across multiple documents. This situation creates a need to first "JOIN" the information in order to create a complete document for analysis. A new difficulty is added to the system, the application must also be able to aggregate information from a variety of sources without a general schema and using different "keys" according to the different sources that are being analyzed.

4.2 Requirements

The developed prototype was conceived having in mind a series of requirements that needed to be followed:

- The results must be produced in appropriate time, the time must not greatly surpass that of the current solution;
- The results must be consistent with the ones from the solution that currently is being used, thus assuring the correct mapping of documents;
- The application must not spend considerably more resources than the current solution.

4.3 Proposed Solution

The proposed solution is based around the use of two steps to process the information, the mapper and reducer. In the first phase the information is merged, meaning that information that was previously dispersed across the database are now in a single document. On the second stage the information that is relevant is selected form that was previously joined. Both these steps make use of a configuration document each where the information about the join rules for the information as well as the transformations that are required from the data.

The prototype of the solution was developed using JAVA 8 and uses MongoDB as a source and final destination for the information. These technologies were chosen because of their relevance in the case being studied. It was developed with as many functionalities as possible and in a way

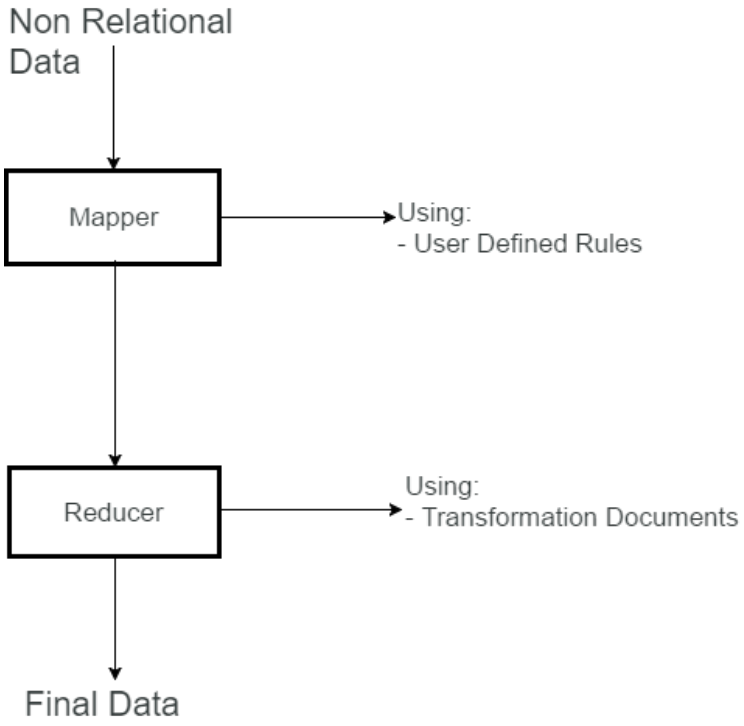

Figure 4.1: Base Application Architecture

that could facilitate any eventual future developments. As described above the prototype works around two configuration files where the rules for the "mapping" and "reducing" stages and a base configuration file for the prototype.

4.4 The Development

During the first phase of the development an evaluation was made for the adaptation of algorithms used in an entity matching problem to the situation previously described. These types of algorithms require a certain degree of information in order for them to be correctly used, so that a more correct match can be made. However, due to the great amount of similarities in the attribute types, the great majority of elements are strings both in the initial and in the final structure, as well as no correlation between the names in the two states of the information. This situation made the usage of entity matching algorithms not viable and a more deterministic solution was devised. The prototype that was developed made use of two configuration files, one for each step of the execution, that are defined by the user in order to perform the mapping and reducer steps, these files were defined so that the knowledge that would be needed about the inner structure of the data is minimized.

Table 4.1: Performance tests

	Developed Application (without generic paths)	Developed Application (with generic paths)	Reference
Records	1404	1404	1404
Documents	5000	5000	5000
Execution time	1m:38s	1m:45s	1m:30s
Memory Used	460MB	500 MB	430MB
CPU used	26%	30%	20%

4.5 Validation and Testing

This prototype will be tested using real world data that has been gathered from a variety of sources. The first step, the mapper, will be tested by a manual analysis of a random sampling of the documents that have been processed to assess the effectiveness of the operation.

The second step, the reducer, is evaluated by comparing the results with the application currently being used by the company. This approach is used to ensure that the application being developed is comparable to the one already used by the company and that it is ready to handle real data.

The tests were performed on a machine with the following characteristics:

- Operative System - Windows 10 Enterprise
- Processor - Intel Core I7-4790 @ 3.6GHz
- RAM - 8 GB

Testing results are present in table 4.1, records represent the number of base elements in the system, in the test case "flows" and documents represent the total number of documents that can possibly be matched in the system. The files used are present in Appendix A and Appendix B.

4.6 Scalability

A big part of an application that deals with Big Data is the ability to scale horizontally. In order to tests this the prototype was deployed using Apache Spark. A test was conducted using two machines running independent instances of the prototype using Kafka streams to feed them information. The end result was an improvement of the processing time of the documents to 1 minute.

This serves as a proof of concept in order to assess that the application can correctly scale horizontally, so that it can be applied to real-world scenarios with greater degrees of information needing to be processed.

4.7 Summary

Data collection is playing an ever an ever increasing role in companies activities and decision making processes. Despite this relevance a problem concerning the data cleaning process, there is currently no process for the automatic mapping of unstructured information. The proposed solution would create an alternative to the current method of creating a different application for each merger that was required.

There are two main components to the proposed solution: mapper and the reducer. The first represents the merging of information from a variety of files into a single one, this makes the transformation of the structure of the information easier. The second step is the transformation of the information itself, this is done by using transformation documents where the correspondences of the different attributes are stored and rules that are extracted from the different documents.

The proposed solution can be more accurately tested and validated by using real world information and by comparing results with the solution currently being used. The following chapters are used to describe in greater detail the development and capabilities of the steps.

Chapter 5

Developed Solution

This chapter describes the development of the mapper stage of the prototype, as well as the difficulties encountered. Additionally it is also described the development of the reducer stage.

5.1 Mapper Stage

The mapper stage is the first of the two main steps in the overall architecture of the prototype. It serves as a way for the relevant information to the problem at hand to be joined in a single document. This operation must be possible as long as the relevant information is present within the database, regardless of the number of documents that are needed to successfully make the join. Additionally, an added concern was considered while the rule document schema was being planned, a conscientious effort was made in order to try and minimize the amount of information, about the inner structure of the information itself, that would be required of the end user for the correct use of the application.

All of the design decisions that will be discussed over this chapter were made to address the problem of mapping unstructured data, to create a prototype that could be used in a real world scenario and in which additional features may be built in future developments.

5.2 Mapper Development

The aim of the first development stage was the creation of an application where a user can define a series of rules in order to correctly merge the relevant documents that are present in a database into a single one. In the early stages of development the rules were viewed as simple correspondences between the different attributes, however a continuous contact and an analysis of the data made it clear that some additional functionalities were required that would add to the complexity of the file the user needs to write. Nonetheless, after an cost\benefit analysis it was decided that the added functionality would bring sufficient improvements to justify the added intricacy.

During this stage of the development was when the limitations to the entity matching algorithms was found, and it was when the use of user-defined documents was chosen to serve as a substitute. The document is written in JSON format and is used to point to the file where the transformations are defined.

5.3 Mapper Functionalities

As described above, the mapper allows the user to define the rules that are used to join the information in the different documents. In addition to this it is possible to utilize regular expressions in order to better select the attributes in the documents. There is also a possibility of using xPath expressions if the attribute being analyzed is an XML document, and there is a need to perform a query on it. In addition to these rules, there is also a need to define the base components that are needed to complete the connection to the database for both the input and output, as seen in 5.1.

```
1   "hostInput":"192.168.1.174",
2   "portInput":27017,
3   "databaseInput":"mxfSPEEDRAIL",
4   "hostOutput":"192.168.1.174",
5   "portOutput":27017,
6   "databaseOutput":"JOINTest",
7   "collectionOutput":"SpeedRail"
```

Listing 5.1: Example Base Configuration

Additionally, the user is not required to perfectly know the structure of the document in question, the path expressions are written without concern if a particular attribute is an array or an object, and it is also possible for a user to use a generic identifier to replace one or more attributes that they do not know. This means that if a user does not know a particular attribute, the application will search its descendants for the next known attribute, the search is performed using a breadth-first algorithm in a number of levels equal to the number of generic names used in a row.

Finally the user is also given the opportunity of transforming attributes that contain XML into JSON and JSON Objects into XML attributes. Check Appendix A for complete documentation.

```
1 {
2   "leftCollection":"Flows",
3   "rightCollection":"Assets",
4   "leftAttribute":["supCode","Flows.%%.%%.%%.pf:Input.%%.id"],
5   "rightAttribute":["supCode","Assets.Id.Id"],
6   "transformation":{
7     "type":"XMLtoJSON",
```

```
8       "attribute":"Flows.Config.Config",
9       "newAttributeName":"ConfigJSON"
10    }
11  }
```

Listing 5.2: Join Rule with Transformation Example

In the example in 5.2 is shown an example of a join rule in a configuration file that makes use of generic paths and a transformation element. In each document join the collections where the documents are located and an array with the attributes that should be used to make the join. it is also possible to observe the possibility of using generic paths in action, when a path to an attribute is defined a user can replace the name of the attribute with a string beginning and ending with "%" the application uses a depth-first search algorithm to search the next defined attribute, the depth searched is the number of consecutive generic paths. Additionally it is also possible to add an optional "pattern" attribute, where a regular expression, or an xPath expression can be defined to better process the attributes. Lastly, an attribute is transformed from an XML document to a JSON attribute, this is defined by defining the type of transformation that is necessary, "XMLtoJSON" or "JSONtoXML", as well as the attribute path and the new attribute location.

5.4 Reducer Stage

The reducer stage is the second of the two main steps in the overall architecture of the prototype. It transforms the information, that has been joined in the previous step, into a structure that is of use for the following analysis of the data. These transformations might be as simple as a selection of attributes that need no more transformation, or there might be the need to convert the elements to another type, such as a timestamp to time epoch or a number into a string. Additionally it is also possible to set conditions to the writing of the elements in the final structure.

5.5 Reducer Development

The development of this stage was focused on creating a step in the application where the information that had been gathered could be transformed. These transformations are defined in a transformation document written in JSON format, which schema was inspired by Apache Avro. During the initial stage of development Apache Avro was considered to define the transformation documents and also be used to perform the transformations themselves, however this tool lacked some vital flexibility and functionalities and a decision was made to create a custom-made parser and schema. This situation shifted the focus of this stage in development from the possible inclusion of machine learning techniques to attempt to infer rules from a sample of configuration files to the creation of a fully functional prototype that can be used as a basis for future developments.

Table 5.1: Possible Operations in the Transformation Documents

Name	Description
Equality	Writes an attribute if two other attributes are equal
Max/Min	Either selects the max/min value of a certain attribute, or selects an attribute that is associated to the max/min value of another attribute
Existence	Checks if an attribute exists, even if its value is null. Only applicable to boolean fields
Append	Converts the selected attributes to strings and appends them using a user-defined separator
byteConversion	Converts the selected attributes to byte strings, each with an independent user-defined size, and appends them

5.6 Reducer Functionalities

As previously described the reducer base functionality is to directly link between the attributes in the intermediate format and the final structure. Additionally, it is also possible to transform the type of the attributes as well as, performing operations on the attributes, in order to better transform the information into the desired format. The possible operations are presented in table 5.1. Check Appendix B for complete documentation.

```
{
    "name":"operation",
    "type":"string",
    "location":["Flows.Config.ConfigJSON"],
    "oldFormat":"object"
},{
    "name":"time_end",
    "type":"long",
    "location":["Flows.LastUpdate.TimeStampInsert"],
    "operation":{
        "type":"max",
        "data":{
            "type":"long"
        }
    }
}
```

Listing 5.3: Excerpt of a Transformation Document

The transformation document was developed having in mind that, while it should allow a user to perform all of the operations that were previously described, it should be simple to write and interpret. In 5.3 it is shown two elements that are present in a transformation document, in the first one a conversion is being made, signaled by the use of the "oldFormat" element, in this case object means that what is required is the name of the child of the node present in the designated location. In the second case the max operation is being used, meaning that the element that is selected is the max that is present in all of the "TimeStampInsert" attributes.

Chapter 6

Conclusions and Future Work

This chapter is used to present a summary of the work that has been developed as well as an analysis of the goal satisfaction and recommendations for possible future work.

6.1 Conclusions

Nowadays, data analysis and collection have an important place for corporations. Even though data was always being collected only recently there developments have taken place that allowed for the processing of this information. This was caused by developments in both the technologies that are used to store information and in the methods that are used to analyze said information.

In the area of information storage the main development was the detachment from a standard that had been used since the 1980's, the relational database model. This model allows for the storage of information with arbitrary complexity but does so by requiring data to follow a strict schema and, even if it is distributed, having a central location where all read/write operations are submitted. With this rigid structure this type of database is unable to cope with the flow of data that can be created or collected by an application. The detachment came in the form of NoSQL databases that do not need a general schema and are focused on the parallelization of the data storage. Whilst these types of databases can provide a great deal of flexibility they sacrifice the consistency that was a staple of relational databases.

This field has seen developments by the developments in parallelization and distributed computing. These allow the analysis of a larger amount of information and of more complex information. Despite these developments there are still some problems that have not been comprehensively addressed, one of these is the data-cleaning process. This step in data processing is characterized by the selection and transformation of the information that has been gathered so that a maximum of knowledge can be extracted from it. Traditionally this is addressed by creating a case to case solution, meaning that each data structure requires a different solution in order to correctly extract the relevant information from it.

In order to address this situation a more general model is proposed based on two main steps: the mapper step and the reducer step. In a first stage all of the relevant information is merged into a single file and, on a second stage, the information is transformed using a combination of rules defined in transformation documents. The former is necessary because the information that is relevant for a user defined structure can be spread across multiple documents. The latter is the transformation of the information itself into a final format. Even with the need to write the transformation documents it would represent an improvement over the current situation, rewriting a document instead of changing the code of the application.

6.2 Goal Satisfaction

The main goal that was set at the beginning of this dissertation, the development of a prototype application that allows for the easier pre-processing of information, was achieved, however it was not in the manner that was initially expected. During the initial phase of development it was discovered that there was a lack of information present in the system for a use of entity-matching techniques. With this in mind a decision was made to instead utilize user-defined documents to define the rules that are to be used in the join and in the transformations.

Despite this, there was still a reduction to the complexity of the data-cleaning process, when there is a change in the data, or a new set of data is analyzed there is only the need to change the rule documents and not the application itself.

The prototype allowed for these assumptions to be tested and verified in a real-world scenario, the documents can be adapted to a variety of data and operations with relative ease.

The tests that were performed by comparing the results of the prototype with the previously used solution were satisfactory and are presented in tables 6.1, 6.3 and 6.2. These tests demonstrate the ability of extracting the same information from the data with less effort than it is currently possible.

6.3 Future Work

Using the result of this dissertation work a series of research can be made in order to better access the usability of the prototype and well build upon it.

Despite using real-world test data all areas of knowledge were not represented meaning that a functionality, or type of operation, might be missing that would make sense in a certain context. Also it would be interesting for a more complete analysis of the suitability of the prototype.

It is also important to note that a series of difficulties encountered during this dissertation work made it inviable to implement some machine learning techniques within the application. There is a possibility that some useful functionalities may be added with the use of this technology, such as the ability of making recommendations for possible transformations based on earlier transformation documents.

Table 6.1: Original Information

Initial Data	
InsertTime	2017-04-13 10:41:32.000Z
supCode	10.20.30.10
timestamp	1492075271.0
machineIdentifier	5379150.0
processIdentifier	4668.0
counter	10405710.0
kafkaRun	0
kafkaSend	true
AssetID	58e7c7f252144e16905991b3 (Inside Config attribute)

Table 6.2: Previous Application Result

Previous Application Result	
InsertTime	1492080092000
supCode	10.20.30.10
_id	58ef430752144e123c9ec74e
info	10.20.30.10:true:0
assetID	58e7c7f252144e16905991b3

Table 6.3: Final Result

Final Result	
InsertTime	1492080092000
supCode	10.20.30.10
_id	58ef430752144e123c9ec74e
info	10.20.30.10:true:0
assetID	58e7c7f252144e16905991b3

Another functionality that can be analyzed is the automation of the transformation step, with enough information there might be the possibility of instead of defining a transformation document, a user would define an information document. Meaning that instead of defining all of the informations that are needed no be done, a user would state the type of information that is required. For example a user would define that needed all of information about a topic and the application could perform the transformations necessary for that to be possible.

Finally, additional tests on the application in order to assess how can it scale as well as optimizations that might become necessary with a prolonged use of the application.

References

[avr] Apache avro™ 1.8.1 documentation. http://avro.apache.org/docs/current/. Accessed: 2017-01-31.

[DG08] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[Dij14] Jean-Pierre Dijcks. Oracle: Big data for the enterprise. 2014.

[GH15] Amir Gandomi and Murtaza Haider. Beyond the hype: Big data concepts, methods, and analytics. *International Journal of Information Management*, 35(2):137 – 144, 2015.

[had] Welcome to apache™ hadoop®! http://hadoop.apache.org/. Accessed: 2017-01-31.

[IBM] Infographic: The four v's of big data | ibm big data & analytics hub. http://www.ibmbigdatahub.com/infographic/four-vs-big-data. Accessed: 2017-01-31.

[kafa] Apache kafka. https://kafka.apache.org/documentation/. Accessed: 2017-05-29.

[kafb] Putting apache kafka to use: A practical guide to building a streaming platform. https://www.confluent.io/blog/stream-data-platform-1/. Accessed: 2017-05-29.

[SAS] What is big data? | sas us. http://www.sas.com/en_us/insights/big-data/what-is-big-data.html. Accessed: 2017-01-31.

[spa] What is apache spark? https://databricks.com/spark/about. Accessed: 2017-06-12.

REFERENCES

Appendix A

Join Rule Documents

In this annex is shown an example of a join rules document and definition of the document's schema.

A.1 Example File

```
{
  "hostInput":"192.168.1.174",
  "portInput":27017,
  "databaseInput":"test_mxfSPEEDRAIL",
  "hostOutput":"192.168.1.174",
  "portOutput":27017,
  "databaseOutput":"JOINTest",
  "collectionOutput":"SpeedRail",
  "joinRules":[
    {
      "leftCollection":"Flows",
      "rightCollection":"StorageServers",
      "leftAttribute":["supCode","Flows.Config.Config"],
      "rightAttribute":["supCode","StorageServers.id.id"],
      "pattern":["","//SMB_CIFS/@id"]
    },{
      "leftCollection":"Flows",
      "rightCollection":"Assets",
      "leftAttribute":["supCode","Flows.%%.%%.%%.pf:Input.%%.id"],
      "rightAttribute":["supCode","Assets.Id.Id"],
```

```
21        "transformation":{
22          "type":"XMLtoJSON",
23          "attribute":"Flows.Config.Config",
24          "newAttributeName":"ConfigJSON"
25        }
26      }
27    ],
28    "ReducerDocumentPath":"C:/Users/pedrorocha/Documents/Dissertation/
        src/main/resources/transformation.json"
29  }
```

Listing A.1: Join Rules Document With Generic Paths Example

A.2 Document Definition

With the exception of the "ReducerDocumentPath" attribute all of the base attributes in the document A.1 are required in every document. The attributes are as follows:

- **hostInput** - IP address of the input database, string;
- **portInput** - the port for the input database, integer;
- **databaseInput** - the name of the input database, string;
- **hostOutput** - IP address of the output database, string;
- **portOutput** - the port for the output database, integer;
- **databaseOutput** - the name of the input database, string;
- **collectionOutput** - the name for the output collection, string;
- **ReducerDocumentPath** - path to the file where the transformation rules are defined, string;

The element "joinRules" is defined as an array of objects, these objects being each one a different rule that can be applied to all the collections present in "databaseInput". The rule object contains both required and optional attributes, the latter ones exist so that some transformations can be made to the information in this point in the application. It is required to use the following attributes:

- **leftCollection** - represents the collection from which the left side of the comparison will be selected, string;
- **rightCollection** - represents the collection from which the right side of the comparison will be selected, string;

- **leftAttribute** - paths to the elements used in the comparison, array of strings;
- **rightAttribute** - paths to the elements used in the comparison, array of strings;

These are used to select the attribute(s) that should be used in the join, there should be the same number of attributes in both sides of the comparison as well as they should be in the same order, the first element is compared to the other first element and so forth. It is also possible to use the attribute "pattern" (string) to define the regular expression or xPath expression to be used on the value of the left attribute (a pattern is assumed a xPath expression when it contains "/").

Finally the "transformation" object is used when attributes need to be transformed to or from JSON/XML, the transformation from JSON to XML converts the selected attribute children into an XML document, the XML to JSON transformation converts the value of the selected attribute, an XML document, into JSON. This object contains three required fields :

- **type** - "JSONtoXML" or "XMLtoJSON";
- **attribute** - path to attribute to change, string;
- **newAttributeName** - new name for the changed attribute, string;

A.3 Schema

This schema file A.2 is used in runtime to check the validity of the rule file.

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "properties": {
        "collectionOutput": {
            "type": "string"
        },
        "databaseInput": {
            "type": "string"
        },
        "databaseOutput": {
            "type": "string"
        },
        "hostInput": {
            "type": "string"
        },
        "hostOutput": {
            "type": "string"
        },
```

```
        "joinRules": {
            "items": {
                "properties": {
                    "leftAttribute": {
                        "items": {
                            "type": "string"
                        },
                        "type": "array",
                        "uniqueItems": true,
                        "minItems":1
                    },
                    "leftCollection": {
                        "type": "string"
                    },
                    "rightAttribute": {
                        "items": {
                            "type": "string"
                        },
                        "type": "array",
                        "uniqueItems": true,
                        "minItems":1
                    },
                    "rightCollection": {
                        "type": "string"
                    },
                    "transformation": {
                        "properties": {
                            "attribute": {
                                "type": "string"
                            },
                            "newAttributeName": {
                                "type": "string"
                            },
                            "type": {
                                "type": "string"
                            }
                        },
                        "type": "object",
                        "required":["attribute","type","
                           newAttributeName"]
```

```
                    }
                },
                "type": "object",
                "required":["leftCollection","rightCollection","
                   leftAttribute","rightAttribute"]
            },
            "type": "array",
            "uniqueItems": true,
            "minItems":1
        },
        "portInput": {
            "type": "integer"
        },
        "portOutput": {
            "type": "integer"
        },
        "transformationDocumentPath": {
            "type": "string"
        }
    },
    "type": "object",
    "required":["hostInput","portInput","databaseInput","hostOutput"
       ,"portOutput","databaseOutput","collectionOutput", "joinRules
       "]
}
```

Listing A.2: Join Rules Schema Document

Appendix B

Transformation Documents

In this annex is shown an example of a transformation document and definition of the document's schema.

B.1 Example File

```
{
    "name":"base",
    "type":"record",
    "fields":[{
        "name":"code_machine",
        "type":"string",
        "location":["supCode"]
    },{
        "name":"insert_time",
        "type":"long",
        "location":["INSERTED"],
        "oldFormat":"date"
    },
    {
        "name":"IPv4_addr",
        "type":"string",
        "location":["IPv4Addr"]
    },{
        "name":"obj_input",
        "type":"string",
```

```
        "location":["Assets","Assets.Id.Id"],
        "operation":{
            "type":"equality",
            "data":{
                "left":["Flows.Config.ConfigJSON.pf:TransferPlainMXF
                    .pf:Input.pf:Asset.id"],
                "right":["Assets","Assets.Id.Id"]
            }
        }
    },{
        "name":"operation",
        "type":"string",
        "location":["Flows.Config.ConfigJSON"],
        "oldFormat":"object"
    },{
        "name":"time_end",
        "type":"long",
        "location":["Flows.LastUpdate.TimeStampInsert"],
        "operation":{
            "type":"max",
            "data":{
                "type":"long"
            }
        }
    },{
        "name":"status",
        "type":"object",
        "location":["Flows.Status.Status"],
        "operation":{
            "type":"min",
            "data":{
                "type":"long",
                "element":["Flows.Status.TimeStampInsert"]
            }
        }
    },{
        "name":"spanned",
        "type":"boolean",
        "location":["Assets","Assets.Spanned"],
        "operation":{
```

```
60              "type":"existence"
61          }
62      },{
63          "name":"append",
64          "type":"string",
65          "operation":{
66              "type":"append",
67              "data":{
68                  "elements":[["supCode"],["Assets","kafkaSend"],["
                        kafkaRun"]],
69                  "separator":";"
70              }
71          }
72      },{
73          "name":"byteConversion",
74          "type":"String",
75          "operation":{
76              "type":"byteConversion",
77              "data":{
78                  "size":[4,3,2,3],
79                  "location":[["Flows._id._id.timestamp"],["Flows._id.
                        _id.machineIdentifier"],["Flows._id._id.
                        processIdentifier"],["Flows._id._id.counter"]]
80              }
81          }
82      }]
83  }
```

Listing B.1: Transformation Document Example

B.2 Document Definition

The document structure is based on Apache Avro with some additional fields B.1. The base structure is based around objects, "records", which required fields are:

- **name** - name of the element in the final document, the name of the first object is ignored, string;
- **type** - type of the object in the final document: int, long, double, string, array, object, array or record, defined with a string;

Table B.1: Required fields in each operation

Operation	Data Field Contents
Equality	"right" and "left" (array of strings) attributes that are used to check if attribute is written or left blank
Max/Min	"type" (string) required, to indicate the type of the attribute (long, int, double), optional "element" (string) used when the element being maximized (or minimized) is different than the one being written
Existence	no "data" field required
Append	"elements" (array of array of strings) is used to indicate the paths of the different attributes that are to be appended and "separator" (string) the separator to be used between the attributes
byteConversion	"size" (array of integer) is used to indicate the byte size that each element will have and "location" (array of array of strings) is used to identify the location of the attributes

When fields are of an array or record type the "fields" (array of strings or objects) attribute becomes mandatory, in the former the type of elements that is allowed in the element is defined, the latter requires an array of objects, that are contained within.

The attribute "location" (array of strings) is used to identify the location of the attribute in the original document and "oldFormat" (string) is used to indicate the previous type of the attribute in the original document, used in type conversions operations.

The "operation" attribute is an object and is used when a more complex operation is needed it contains a required field (string), showing the type of operation that is needed to perform (equality, max/min, append or byteConversion) and an optional "data" attribute (object) that can contain several attribute, table B.1 shows the required fields for the different type of operation.

It was decided against using a schema file to validate the transformation document due to the great amount of optional fields that exist.